DTIC FILE COPY UNLIMITED

AD-A216 297

RSRE MEMORANDUM No. 4299

ROYAL SIGNALS & RADAR ESTABLISHMENT

COMPUTING OPTIMUM ROUTES ACROSS TERRAIN USING DAP

Authors: R Haynes, C Packer, J B G Roberts & P Simpson

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
R S R E MALVERN,
WORCS.

90 01 02 052

RSRE MEMORANDUM No. 4299

UNLIMITED

0055565

CONDITIONS OF RELEASE

BR-111965

*************************** U

*************************** Y

Reports quoted are not necessarily available to members of the public or to commercial *organisations.*

Royal Signals and Radar Establishment

Memorandum 4299

COMPUTING OPTIMUM ROUTES ACROSS TERRAIN USING DAP

R Haynes, C Packer, J B G Roberts, P Simpson

Royal Signals and Radar Establishment
St Andrews Road, Great Malvern, Worcs WR14 3PS United Kingdom

July 1989

SUMMARY

Dynamic programming techniques provide quantitative ways for determining optimum routes across complex terrain for various assumed categories of vehicle, taking account of gradients, the presence or absence of roads and obstacles such as rivers, urban areas, woods etc. We have investigated the effectiveness of a highly parallel SIMD computer. (Mil-DAP) to this problem. We conclude that, although impressive computation speeds can be obtained on high resolution digital maps with Mil-DAP, the conflict between exploiting high order SIMD parallelism and minimising the number of necessary computation operations is such that in this case Mil-DAP does not command a dominating advantage over a conventional uniprocessor.

CONTENTS:

1 INTRODUCTION:

The calculation of optimal paths for troop and vehicle movements over complex terrains in mission management represents an important military problem. The object of this study was to discover to what extent the power of a highly parallel SIMD (Single Instruction Multiple Data) processor such as the DAP is applicable to the route optimising problem. In particular could massive parallelism economically compete with the algorithmic efficiency gains available to a single sequential processor using subtle mechanisms to steer the node updating process?

The architecture of the DAP is well documented [1] and is not covered by this report, but is particularly well suited to calculations involving 2-D arrays such as maps based on a square grid. It was hoped that by applying the massive parallelism of the DAP to this kind of problem that large improvements in performance could be obtained. Comparisons in performance between the DAP and a sequential machine were made by using Moore's and D'Esopo's shortest path algorithms running on a VAX 8600. Some reference is made to a complementary piece of research by Hislop [2] currently investigating the use of Transputers for the same class of problem.

2 ROUTE FINDING ALGORITHMS [3,4].

Dynamic Programming is a general term covering all the algorithms we use. The first algorithm described is a simple unguided algorithm, the others use queuing strategies to order the visiting of nodes in ways intended to reduce the number of updates necessary.

2.1 Simple Dynamic Programming [5,6].

Dynamic programming is an optimisation technique in which a decision is broken down into a series of small steps. In a network of points, for example a map of spot heights, the transitions between two distant points can be broken down into a series of moves between adjacent points. Using the Principle of Optimality [7,8] - "Any sub path of the optimal path is itself optimal" - the problem can be broken down into progressively smaller steps establishing optimal transitions between adjacent points. Costs are assigned to moves between all pairs of adjacent points, which can be a function of various parameters e.g absolute differences in height, the presence or absence of a road linking the points, or a combination of these and other factors. The optimal route between distant points will be the set of moves between adjacent points for which the sum of costs for all the moves is a minimum. Each point is assigned a cumulative cost which represents the minimum cost of getting to that point from the start. In dynamic programming each point is referred to as a Node and a move between two adjacent points as a Link. The minimum cumulative costs are established iteratively.

The dynamic programming algorithm proceeds as follows:-

Initially all the node costs are set to infinity except the start node cost which is set to zero. The nodes are selected in a fixed arbitrary sequence and a check is made to see if the cumulative cost at each selected node can be reduced by taking a path to it from any of its adjacent nodes. If the adjacent node cost plus link cost from that node is less than the current cost of the selected node this node's cost is updated with the smaller value. The selection process is continued until all the node costs cease changing at which point the algorithm terminates.

Tab - Transition(link) cost between 2 adjacent nodes a and b.

Ca^k - Cost of node a after k iterations.
Cb^k - Cost of node adjacent to node a.
n - Number of nodes adjacent to node a.

$$C^o{}_{start} = 0$$
$$C^o{}_{rest} = \text{infinity}$$
$$Ca^k = \min\{ C_a{}^{k-1}, \{ Cb_z{}^{k-1} + Tab_z\}\} \quad z = 1\ldots n$$

The algorithm for a network of n nodes expressed in pseudo FORTRAN is:-

SNC - Selected node cost
ANC - Adjacent node cost
LC - Link cost between selected node and adjacent node
SN - Selected node

```
DO 10 i=1,n
     NC(i) = infinity
10 CONTINUE

NC(start) = 0

15 CONTINUE

     DO 20 i = 1,n

          DO 30 For each node j adjacent to node i
               IF ([ANC(j) + LC(j)].LT.SNC(i))
                    THEN SNC(i) = ANC(j) + LC(j)
                         ALSO set a flag(U) to indicate an update.

          30  CONTINUE

     20 CONTINUE

IF flag(U) set GOTO 15

END
```

An worked example can be found in Appendix A.2.

In this dynamic programming algorithm the node selected looks at its adjacent nodes and updates itself if a lower cost path is found (looking in); some other algorithms select a node and update adjacent nodes (looking out).

2.2 Moore's algorithm [9]

Moore's algorithm is a faster sequential dynamic programming algorithm, the difference being in the order the nodes are visited, which is determined by a queuing mechanism. Initially, as in basic dynamic programming, the node costs are all set to infinity and the start node cost to zero. However a queue is set up onto which the start node is initially placed. Nodes are selected and removed for examination, always from the front of the queue, whilst other are added to the queue as follows. If the selected node cost plus link cost to an adjacent node is less than the present cost at that adjacent node, then the adjacent node is updated with this smaller value. If this node cost is changed and is not

already somwhere on the queue, it is added to the end of the queue. This is repeated for all the nodes that are adjacent to the selected node. This "looking out" process continues until the queue becomes empty at which point the algorithm terminates.

$$C^{o}_{start} = 0$$
$$C^{o}_{rest} = \text{infinity}$$
$$Cb^{k} = \min (Cb^{k-1}\ Ca^{k-1} + Tab)$$

For a network of n nodes the algorithm, expressed in pseudo FORTRAN, is :-

```
DO 10 i=1,n
      NC(i) = infinity
10 CONTINUE
NC(start) = 0
Front = 1
Queue(Front) = start node

15 CONTINUE
      SN = Queue(Front)
           Front = Front + 1

                DO 20 each node j adjacent to node SN

                     IF ([SNC + LC(j)].LT.ANC(j))
                          THEN [ANC(j) = SNC + LC(j)]
                               and IF adjacent node not on Queue
                                         THEN put on Queue(End)
                                              and End = End + 1

20  CONTINUE

IF [Queue empty] END,  ELSE GOTO 15

END
```

A worked example is given in Appendix A.3.

2.3 D'Esopo's algorithm [10,11].

D'Esopo's sequential algorithm is identical to Moore's algorithm except in the way the queue is ordered. In Moore's algorithm nodes are always put on the end of the queue whereas in D'Esopo's algorithm a node is put at the end only if it has not already been on the queue at some earlier iteration. If it has, the node is put at the front of the queue. This has the effect of giving priority to nodes which have been modified previously , thus clearing up back-tracking quickly.

$$C^{o}_{start} = 0$$
$$C^{o}_{rest} = \text{infinity}$$
$$Cb^{k} = \min (Cb^{k-1}\ Ca^{k-1} + Tab)$$

For a network of n nodes the algorithm is expressed in pseudo FORTRAN.

```
DO 10 i=1,n
      NC(i) = infinity
10 CONTINUE NC(start) = 0
Front = 1
Queue(Front) = start node

15 CONTINUE
      SN = Queue(Front)
           Front = Front + 1

           DO 20 each node j adjacent to node SN

                IF {[SNC + LC(j)].LT.ANC(j)}
                     THEN [ANC(j) = SNC + LC(j)]
                           and IF adjacent node not on Queue
                                    THEN put on Queue(End)
                                         and End = End + 1

           20 CONTINUE

IF [Queue is empty] END,  ELSE GOTO 15

END
```

A worked example is available in Appendix A.4.

3 IMPLEMENTATIONS

We now describe:-

(a) Parallel implementation of simple Dynamic Programming on Mil-DAP.
(b) Improved DAP algorithm (processing the active blocks and immediate update).
(c) Moore's sequential algorithms on VAX 8600.
(d) D'Esopo's sequential algorithm on VAX 8600.

The improvements to Simple Dynamic Programming used here on DAP do not include the queuing strategies of Moore's and D'Esopo's we have used on the VAX implementations as these are intrinsically incompatible with the SIMD parallelism of the DAP. MIMD parallelism for these algorithms is being investigated separately [2].

The problem taken was to find the shortest path from a starting point to all other points on a map. The map was a 256*256 map of 50m spaced spot heights of a section of the Isle of Wight (I.O.W), together with an overlay containing information on the location of trees, buildings, surface water, railway tracks and roads. The spot height data is referred to as the Terrain and the overlay as the Culture. The advantage in using the DAP was one could work on blocks of data, so the I.O.W map being made up of 256*256 points (nodes) was divided up into an square grid of 64 (8*8) blocks each containing 1024 (32*32) nodes.

We have also compared the performances of the same algorithms on a "pathological" or worst case terrain in which the optimal solution is a zigzag path from one corner to another.

3.1 Direct parallel implementations of simple Dynamic programming using DAP

The parallel implementation of dynamic programming is essentially the same as the sequential version described earlier, having the same start conditions and basic steps, but instead of checking one node at a time it simultaneously checks a block of 1024 nodes.

The DAP can manipulate large arrays in single steps so the inner loop for one 32*32 block only, neglecting interactions between blocks (for simplicity), of the route optimisation programming in DAP FORTRAN is.

```
INTEGER*2  DISP(,)       Array of old node costs.
INTEGER*2  NEWS(,)       Array containing old and new updated node costs.
INTEGER*2  BLK(,)        Array of adjacent node costs + link costs.
INTEGER*2  DN(,)         Array of link cost for northward move.
LOGICAL    MASK(,)       Array of old costs to be updated.
CHARACTER ROUTE (,)      Array of back markers so the route can be traced
                         back to the start node.

C---A block of nodes looks to its southern neighbours with the link
C---costs added, where this value is less than the current value
C---the node is updated.

10    CONTINUE

C---  Shift the old node costs to the north and store these in BLK(,)

      BLK(,) = SHNP(DISP(,))

C--- Any node effectively shifted from beyond the southern edge of
C--- the network is given an "infinite" cost, BIG.

      BLK(32,) = BIG

C--- The block of link costs for a southward move is added to the
C--- nodes from the south.

      BLK(,) = BLK(,) + DN(,)

C---  A  logical overlay is made of nodes that require updating.

      MASK(,) = BLK(,).LT.NEWS(,)

C--- This overlay is placed over the matrix of old node costs and those
C--- requiring update are updated with the cost.

      NEWS(MASK(,)) = BLK(,)

C---  A back marker is then set to show in which direction the cheaper
C---  adjacent node was found.

      ROUTE(MASK) = 'N'
```

This process is repeated for the eastern, southern and western nodes.

After all the moves have been checked:-

```
C--- If any nodes have been updated set a flag (ACTIVE)

     IF (ANY(NEWS(,).NE.D'SP(,))) ACTIVE = .TRUE.

C--- Overwrite the old cost array with the new cost array.

     DISP(,) = NEWS(,)

C--- If any node has been updated repeat process, else terminate.

     IF (ACTIVE) GOTO 10

     END
```

The pseudo DAP FORTRAN below shows how this inner loop is applied to a network of n*n blocks, the blocks contain 32*32 nodes with each node having 4 links.

```
DO 10 i=1,n
      DO 10 j=1,n
           NC(,,i,j) = infinity
10 CONTINUE

NC(start) = 0

15 CONTINUE

      DO 20 i = 1,n
           DO 20 j = 1,n

                INNER LOOP GIVEN ABOVE

           20  CONTINUE

IF flag(U) set GOTO 15

END
```

The full DAP FORTRAN code, including block-block interactions, for this section of the program can be found in Appendix D.1. The worked example in Appendix A.2 may also be used to show a parallel implementation, as the algorithm performs the same steps as the sequential version.

<u>Initial calculation the link (move) costs.</u>

Due to limitations of array store, movement between adjacent nodes was restricted to north, south, east and west moves; diagonal moves were not included thus permitting only four links per node instead of eight. For each set of links the costs were stored in a 256*256 array labelled DN, DS, DE and DW for the north, south, east and west moves respectively. The I.O.W map contains several features that affect the link costs, these are:-

Distance(D)	Cost of travelling 50m
Height(H)	Cost of ascending/descending 1m
Trees(T)	Cost of negotiating wooded areas
Buildings(B)	Cost of negotiating built up areas
Water(W)	Cost of negotiating water
Railway tracks(RW)	Cost of negotiating railway tracks
Roads(R)	Cost of negotiating un-roaded areas

The importance of these features differed with each type of vehicle, four different types were considered and the cost weightings are given in Table 1.

The calculation proceeds as follows:-

Firstly the absolute height difference between the selected node and the adjacent node being moved to is calculated; this is then multiplied by a height factor (H). Added to this value is a distance factor (D) which represents the horizontal distance between the selected node and the adjacent node. If any culture features such as trees (T) or water (W) occur at the adjacent node a value representing the difficulty of travelling across this features is added.

$|h_a - h_b|$ = Absolute height difference between node a and node b
R- = No road

Tab = $H*|h_a - h_b| + k$

where $k = D + T + B + W + RW + R$

Table 1.

VEHICLE TYPE

Terrain feature	TANK	FOOT	CAR	AMPHIBIAN
Distance	5	8	2	4
Height	1	1	1	1
Trees	40	2	50	60
Buildings	150	150	50	150
Water	25	100	50	0
Railway	5	4	50	40
Not Roads	1	1	50	5

The costs were chosen to give a marked difference in the optimal route selected for the various transport types and the user can readily select alternative values at run time, if required. Photographs showing the route selected by a tank and a car are in Appendix C.1 and C.2 respectively. The photographs, Appendix C.3 and C.4, show a cost contour map that represents the final node costs for a tank and a car respectively. The contours are close together where a high transitional cost occur, such as crossing the sea or travelling off road in a car, and further apart where a low transition cost occurs.

It should be noted that as a consequence of not including diagonal moves it was necessary to expand the width of roads to 2 pixels, so that vehicles could maintain a path along a diagonal road without having to stray onto the surrounding terrain. This was achieved by extending all roads by one pixel in a northerly direction.

3.2 Improvements to simple Dynamic Programming on DAP.

Two alterations were made to the simple dynamic programming algorithm in an attempt to improve the execution time of the algorithm.

3.2.1 Processing active blocks only.

It was seen that a wavefront of update activity propagated away from the start point as the optimisation proceeded, with the trailing edge of the wave leaving the area when updating is complete. This can be seen in the photographs in Appendix C.5 and C.6. Initially checks on blocks of nodes not part of the wave of activity were made, i.e. those not requiring any updating at that stage. The algorithm was modified so that only blocks of nodes in the wave area were checked thus saving work. The overall effect was to concentrate work where it was most required. This on average gave a factor of 4 speed up, over the simple dynamic programming algorithm using the I.O.W data. Detailed timing figures are given in Appendix B.1. A copy of the DAP FORTRAN code for the algorithm section, containing this modification and the calculation of the movement costs is given in Appendix D.1.

3.2.2 Immediate overwriting of node cost.

Also attempted was the updating of the node costing's at the first opportunity to see if this improved the speed of the algorithm, however negligible differences in running time were observed (typically 1%). The principle is demonstrated in Appendix A.5.

3.2.3 Further ideas discussed.

It was possible that improvements might be made by using a queue to order the update of blocks of nodes but it was thought that the added implementation time of maintaining a relatively small queue would be greater than any savings made.

A parallel implementation of D'Esopo's algorithm in which all nodes on the queue where considered simultaneously was thought to be impractical due to the extra time that would be needed to sort the node information before each update, the current DAP algorithm simply updates a block of adjacent nodes.

3.3 Moore's and D'Esopo's algorithms running on a VAX 8600.

Both algorithms were run on a square network of nodes of size NA and to simplify the programming only the absolute difference in height was used in calculating the link costs. The main programming loop for each algorithm is detailed below.

3.3.1 Moore's algorithm

```
INTEGER*4 A   Array of numbers for the first link cost of each node i.e.
              each node R has four link costs and A(R) points to the
              position where the  first of these costs occur in array T. As
              the links are stored sequentially the last link of node R
              occurs at A(R+1)-1.
INTEGER*4 B   Array contains the list of nodes that are adjacent to any
              given node.
INTEGER*2 T   Array containing all the link costs between nodes.
INTEGER*4 L1  Array of back nodes i.e. the node you move to when trace the
              path back to the start node.
INTEGER*4 L2  Array of node costs.
INTEGER*4 Q   This array is the queue of nodes to be selected.
          R   The current node
          NA  Number of nodes
         QE = NA + 1
         QS = 0
         QD = 0
10     CONTINUE
C---   QU is the pointer for the start of the circular queue QS is the
C---   start and NA is the maximum length of the queue.
       QU = JMOD(QS,NA)+1
       QS = QS+1
C---   Select node from front of queue and store in R.
       R = Q(QU)
C---   Reverse the sign of the back node to indicate that it is no
C---   longer on the queue
       L1(R) = -L1(R)
C---   From the first link A(R) to the last link A(R+1)-1
       DO 20 I=A(R),A(R+1)+1
C---   Sum the selected node cost and the link cost TB = L2(R) + T(I)
C---   Let P contain the adjacent node
       P = B(I)
C---   Check whether the adjacent node cost is less than TB.
       IF (L2(P).LT.TB) GOTO 90
C---   The adjacent node needs updating and putting on the queue if it
C---   is not already on there.
C---   If the back node value is negative the node is already on the
C---   queue
       IF (L1(P).LT.0) GOTO 20
C***********************************************************************
C---   Put the node on the end of the queue Q, QE is the end marker.
       QD = QD + 1
       QE =
       JMOD(QD,NA)+1
       Q(QE) = P
C---   Set the back node value, and set negative to indicate is presence
C---   on the queue
C***********************************************************************
20     L1(P) = -R
C---   Update the cost of the adjacent node
       L2(P) = TB
90     CONTINUE
C---   Repeat whole process until the queue is empty
C---   If the start pointer QU is pointing at the current end of the
C---   queue QE, then the optimisation is complete so END.
       IF (QU.GT.(QE)) GOTO 100
       GOTO 10
100    END
```

3.3.2 D'Esopo's algorithm

The only difference between D'Esopo's algorithm and Moore's is in the section where nodes are placed on the queue. The D'Esopo version is given below.

```
      IF (L1(P).LT.0) GOTO 20

C****************************************************************************
C---  If the node has at some stage been on the queue put the node to
C---  the front of the queue else put it on the end
      IF (L1(P).LT.(NA+1)) GOTO 15
C---  Put node at end of queue
      QD = QD + 1
      QE=JMOD(QD,NA)+1
      Q(QE) = P
      GOTO 20
C---  Put node on front of queue
15    QS = QS - 1
      QU = JMOD(QS,NA)+1
      Q(QU) = P
C---  Set the back node value, and set negative to indicate is presence
C---  on the queue
C****************************************************************************

20    L1(P) = -R
```

Copies of the FORTRAN code for Moore's and D'Esopo's algorithms are shown in Appendices D.2 and D.3 respectively.

4 COMPARISONS OF PERFORMANCE.

Comparisons between the best SIMD algorithm on DAP and the Moore and D'Esopo sequential algorithms on the VAX 8600 were made to assess the advantages, if any, of using a SIMD parallel machine to implement route optimisation algorithms.

Two types of terrain maps were used for the comparison benchmarks, the I.O.W terrain data excluding the "culture" and a synthetic worst case terrain. Both were variable in size so results could be taken for different sized problems. In all cases the problem was to find the optimal routes from a start node to all other nodes in the network.

The Isle Of Wight (terrain only) data.

This was chosen as a simple practical example for path finding. The original data was made up of 256*256 50metre spaced spot heights of the I.O.W. and the optimal routes were those that involved the least variation in vertical height. The data was laid out in a symmetrical grid pattern. Each node having four adjacent nodes to which it was linked except for nodes at the edge of the network, the link cost were the absolute difference in height between the adjacent nodes. The networks were of variable size, so the data was selected, as a block, expanded out from the top left hand corner of the original 256*256 map. In all cases the starting point was the top left hand corner of the network (Node 1).

It was found, for this data, that the simple Dynamic Programming algorithm run on the DAP was 2 to 3 times faster that D'Esopo's algorithm and this was in turn about 5 times faster than Moore's algorithm, both run on a VAX 8600. The full results and a graph can be found in Appendices B.2.1 and B.3 respectively.

Worst case problem.

This network was chosen for its difficulty in optimising all the routes. It is essentially a valley that starts in the top left-hand corner and zigzags down to the bottom right-hand corner. The "map" was of variable size, up to 65536 nodes, and as before each node had 4 adjacent nodes, therefore 4 links for each node.

Again the link costs were calculated as being the absolute difference in height between adjacent nodes. As can be seen the bottom of the valley was all of height 1 metre and the ridge was all of height 2 metres giving a maximum link cost of 1.

Using this network it was found that D'Esopo's algorithm was about twice as fast as Moore's and about 3 to 4 times faster than the DAP algorithm. Full results and a graph can be found in Appendices B.2.2 and B.3 respectively.

Calculating the transition costs.

In addition to the iterative dynamic programming calculation we have discussed, in practice we need initially to compute the link (transition) costs between each pair of adjacent points as a function of vehicle and terrain characteristics, which may change with time as roads become blocked etc. This task is well suited to a SIMD machine such as Mil-DAP which in typical cases (for a 256*256 map including culture) took 3mS to compute the cost compared with 3s on the 8600. However the advantage of parallelism an this phase of the problem is unimportant since it represents a small fraction of the total computation time.

5 CONCLUSIONS

The key results are embodied in the table of execution times (in seconds) below, extracted from Appendix B.

Problem	Number of Nodes	Best SIMD algorithm on DAP (Active block simple Dynamic programming)	Best sequential algorithm on VAX 8600 (D'Esopo)	DAP/VAX advantage
I.O.W	16K(128*128)	1.05	2.12	2.0
Terrain	65K(256*256)	6.4	19.5	3.0
Zigzag	16K	23.6	6.2	0.26
Valley	65K	204.6	69.5	0.34

Although these timings clearly depend on the terrain characteristics, and also vary a few percent with the choice of starting point, the message is fairly clear. For real terrain data, the DAP speed is only a small factor better than the 8600, this factor not changing greatly with the scale of the problem. The reason for this is evident from the detailed data given at B.3: the 8600 needs to do two to three orders of magnitude less computation than the parallel machine by virtue of its extremely efficient algorithm. The D'Esopo queuing strategy concentrates the processing activity precisely on those nodes whose cumulative costs are on an advancing wavefront of change, ensuring that their iterations are brought to completion before unnecessary attention is paid to temporary knock-on effects away from the wavefront. SIMD parallelism can only feasibly deal with or ignore blocks of (in this case 1024) nodes which prevents the width of the wavefront being minimised.

For the artificial zigzag valley problem the sequential algorithm advantage is maximised so much that the 8600 outpaces Mil-DAP by a factor of 3.

It appears therefore that the use of a somewhat specialised SIMD machine is not generally justified for this class of problem: Although it offers two or three times the speed of a conventional processor, this would not normally be enough to discourage the use of a standard machine, which in any case is solving large (256*256) problems in under 20s.

The interesting issue remains, as to whether a MIMD machine could share the work of highly efficient algorithms like D'Esopo's between a number of processors such a transputers, thereby combining the advantages of the algorithm efficiency and parallelism. We are currently pursuing this line with some expectation of success in conjunction with A.D.Hislop at Southampton University.

Other SIMD work on this problem.

Bitz and Kung [12] have applied the Carnegie Mellon University Warp linear systolic array machine to the path planning problem. Their Dynamic Programming algorithm involves unconditional application of repeated iterations to update the node costs until the values cease to change. In Warp, the nodes are fully raster scanned alternately in each direction in contrast to the conditional 2-D block processing we use on Mil-DAP. Speed comparisons are made difficult because the convergence time is data dependent and because the Warp work allowed diagonal as well as rectangular internode transitions. In terms of add-and-compare operations per second, Mil-DAP seems to be achieving comparable speeds. (We estimate 6.4 MOP/s on Warp and about 17 effective MOP/s on Mil-DAP where "active block" strategy saved a factor of about 4 in the OP/s needed). The conclusion reached with Mil-DAP therefore still stand: that for this particular optimisation problem for which an extremely efficient sequential strategy has been devised, the restricted algorithm freedom of a SIMD machine almost suppresses the normal advantages of parallelism.

6 REFERENCES

[1] P Simpson, J B G Roberts and B C Merrifield, 'Mil-DAP: its architecture and role as a real time airborne digital signal processor' AGARD Conf Proc No 380, Lisbon 20-23 May 1985, pp 23-1 - 23-18.

[2] Hislop, A.D.: "Least Cost Path Finding Algorithms for Digital Maps", University Of Southampton, Dept. of Electronics and Computer Science, 1988.

[3] Dreyfus, S.E.: "An Appraisal of Some Shortest-Path Algorithms", Operations Research, 17, 1969, pp. 395-412.11

[4] Pierce, A.R.: "Bibliography on Algorithms for Shortest Path, Shortest Spanning Tree, and Related Circuit Routing Problems (1956-1974)", Networks, 5, 1975, pp. 129-149.

[5] Denardo, E.V.: "Dynamic Programming, Models and Applications", Prentice Hall Inc., New Jersey, 1982.

[6] Lawler, E.L. : "Combinational Optimisation: Networks and Matroids" Holt, Rinehart and Winston, London, 1976.

[7] Bellman, R.E: "Dynamic Programming," Princeton University Press, Princeton, N.J. 1957.

[8] Dixon, L.C.W: "Nonlinear Optimisation", The English Universities Press Limited 1972.

[9] Moore, E.F.: "The Shortest Path Through a Maze", Proceedings of the International Symposium on Theory of Switching, 1957, pp. 285-292.

[10] Pape, U.: "Implementation and Efficiency of Moore-Algorithms for Shortest Route Problem", Mathematical Programming 7, 1974, pp 212-222.

[11] Pollack, M., Wienbenson: "Solution of the Shortest-Route Problem - A Review", Operations Research, 8, 1960, pp. 224-230

[12] F.Bitz, H.T.Kung: "Advanced Algorithms and Architectures for Signal Processing II", SPIE Vol.826, 1987, pp 215-222.

APPENDIX A: Simple examples of using shortest path algorithms.

A.1 The network of nodes and link costs.

The move costs are derived from absolute difference in height between interconnecting points (Nodes).

```
Node numbers and move (Link) costs
  N7-----1-----N8-----2-----N9
  |            |            |
  |            |            |
  4            2            1
  |            |            :
  |            |            |
  N4-----3-----N5-----5-----N6
  |            |            |
  |            |            |
  12           6            2
  |            |            :
  |            |            |
  N1-----3-----N2-----1-----N3
```

A.2 Dynamic Programming.

Start node = 1

no. of steps	Node Number 1	2	3	4	5	6	7	8	9
Initial	0	inf	inf	inf	inf	inf	inf	inf	inf
1	0	3(1)	inf	12(1)	inf	inf	inf	inf	inf
2	0	3	4(2)	12	9(2)	inf	16(4)	inf	inf
3	0	3	4	12(1/5)	9	6(3)	16	11(5)	inf
4	0	3	4	12	9	6	12(8)	11	7(6)
5	0	3	4	12	9	6	12	9(9)	7
6	0	3	4	12	9	6	10(8)	9	7
7	0	3	4	12	9	6	10	9	7

The process terminates at step (7) since none of the node costs have been updated.

In the sequential version of this algorithm each step represents the systematic update of 9 nodes. The parallel implementation of dynamic programming performs the update on all 9 nodes simultaneously.

The route can be traced back from any node to the start node by using the Back node numbers in the brackets e.g the optimal route from node(9) to node(1), the start, is to go nodes 9,6,3,2,1 in that order.

A.3 Moore's algorithm.

Each step represent the removal of one node from the queue.

Start node = 1

step/ node	1	2	3	4	5	6	7	8	9	Queue
initial	0	inf	inf	inf	inf	inf	inf	inf	inf	1
1 / 1	0	3(1)	inf	12(1)	inf	inf	inf	inf	inf	2,4
2 / 2	0	3	4(2)	12	9(2)	inf	inf	inf	inf	4,3,5
3 / 4	0	3	4	12	9	inf	16(4)	inf	inf	3,5,7
4 / 3	0	3	4	12	9	6(3)	16	inf	inf	5,7,6
5 / 5	0	3	4	12	9	6	16	11(5)	inf	7,6,8
6 / 7	0	3	4	12	9	6	16	11	inf	6,8
7 / 6	0	3	4	12	9	6	16	11	7(6)	8,9
8 / 8	0	3	4	12	9	6	12(8)	11	7	9,7
9 / 9	0	3	4	12	9	6	12	9(9)	7	7, 8
10 / 7	0	3	4	12	9	6	12	9	7	8
11 / 8	0	3	4	12	9	6	10(8)	9	7	

Terminates as the queue becomes empty.

A.4 D'Esopo's algorithm.

Each step represent the removal of one node from the queue.

Start node = 1

step/ node	1	2	3	4	5	6	7	8	9	Queue
initial	0	inf	inf	inf	inf	inf	inf	inf	inf	1
1 / 1	0	3(1)	inf	12(1)	inf	inf	inf	inf	inf	2,4
2 / 2	0	3	4(2)	12	9(2)	inf	inf	inf	inf	4,3,5
3 / 4	0	3	4	12	9	inf	16(4)	inf	inf	3,5,7
4 / 3	0	3	4	12	9	6(3)	16	inf	inf	5,7,6
5 / 5	0	3	4	12	9	6	16	11(5)	inf	7,6,8
6 / 7	0	3	4	12	9	6	16	11	inf	6,8
7 / 6	0	3	4	12	9	6	16	11	7(6)	8,9
8 / 8	0	3	4	12	9	6	12(8)	11	7	7,9
9 / 7	0	3	4	12	9	6	12	11	7	9
10 / 9	0	3	4	12	9	6	12	9(9)	7	8
11 / 8	0	3	4	12	9	6	10(8)	9	7	

Terminates as the queue becomes empty.

A.5 Modified dynamic programming.

__With immediate update of costs.__

Start node = 5

Direction of check	Node Number 1	2	3	4	5	6	7	8	9
Initial	inf	inf	inf	inf	0	inf	inf	inf	inf
North	inf	6(5)	inf	inf	0	inf	inf	inf	inf
East	9(2)	6	inf	3(5)	0	inf	inf	inf	inf
South	9	6	inf	3	0	inf	7(4)	2(5)	inf
West	9	6	7(2)	3	0	5(5)	7	2	4(8)
STEP 1	9	6	7	3	0	5	7	4	

__Without immediate update of costs.__

Start node = 5

Direction of check	Node Number 1	2	3	4	5	6	7	8	9
Initial	inf	inf	inf	inf	0	inf	inf	inf	inf
North	inf	6(5)	inf	inf	0	inf	inf	inf	inf
East	inf	inf	inf	3(5)	0	inf	inf	inf	inf
South	inf	inf	inf	inf	0	inf	inf	2(5)	inf
West	inf	inf	inf	inf	0	5(5)	inf	inf	inf
STEP 1	inf	6	inf	3	0	5	inf	2	inf

APPENDIX B: The results of the algorithm comparisons.

B.1 The modifications to the DAP algorithm.

TL - Start point in top left corner, co-ordinates (1,1)
C - Start point in centre, co-ordinates (128,128)
1 - Tank 2 - Foot soldier 3 - Car 4 - Amphibious vehicle
T - Terrain (height difference only)
Comparisons - Blocks of 1024 nodes that are checked to see if any of the nodes in that block require updating.
Alterations - Blocks of nodes that are selected for updating, there are 1024 nodes in the check with four links so 1024*4 operations are made.

Algorithm	Start Point	Type of vehicle	Time in seconds	Number of comparisons	Number of Alterations
Dynamic	TL	1	18.4	32320	32320
only	TL	2	18.4	32320	32320
	TL	3	19.5	34176	34176
	TL	4	18.4	32192	32192
	TL	T	27.4	47936	47936
	C	1	9.5	16576	16576
	C	2	9.4	16512	16512
	C	3	10.9	19072	19072
	C	4	9.8	17216	17216
	C	T	21.3	37248	37248
Average	-	-	16.3	22835	22835
Dynamic	TL	1	14.1	25408	25408
with	TL	2	14.1	25408	25408
immediate	TL	3	15.2	27392	27392
cost	TL	4	14.6	26304	26304
update	TL	T	20.6	37184	37184
	C	1	7.6	13760	13760
	C	2	7.6	13760	13760
	C	3	8.6	15488	15488
	C	4	7.9	14272	14272
	C	T	16.3	29376	29376
Average	-	-	12.7	22835	22835
Dynamic	TL	1	2.9	32320	4455
with	TL	21	2.9	32320	4489
active	TL	3	4.8	34176	7709
blocks	TL	4	3.5	32192	5512
	TL	T	6.3	47936	10113
	C	1	2.7	16576	4331
	C	2	2.6	16512	4279
	C	3	4.1	19072	6727
	C	4	3.1	17216	5060
	C	T	5.9	37248	9499
Average	-	-	3.9	28557	6217
Dynamic	TL	1	3.6	25408	5958
with	TL	2	3.6	25408	5972
immediate	TL	3	4.9	27392	8176
update &	TL	4	3.9	26304	6388
active	TL	T	5.6	37184	9363
blocks	C	1	2.7	13760	4621
	C	2	2.7	13760	4620
	C	3	3.6	15488	6240
	C	4	2.9	14272	5007
	C	T	4.9	29376	8239
Average	-	-	3.8	22835	6458

B.2.1 Algorithms run using the I.O.W terrain data.

MR - Moore's algorithm
DS - D'Esopo's algorithm
DAP - Dynamic algorithm with active block (on the DAP)
Comparisons - Nodes checked for possible updating, (for the DAP this includes all the nodes in the block).
Alterations - Nodes that are updated, (for the DAP all the node in the selected blocks are included even if not all are changed).

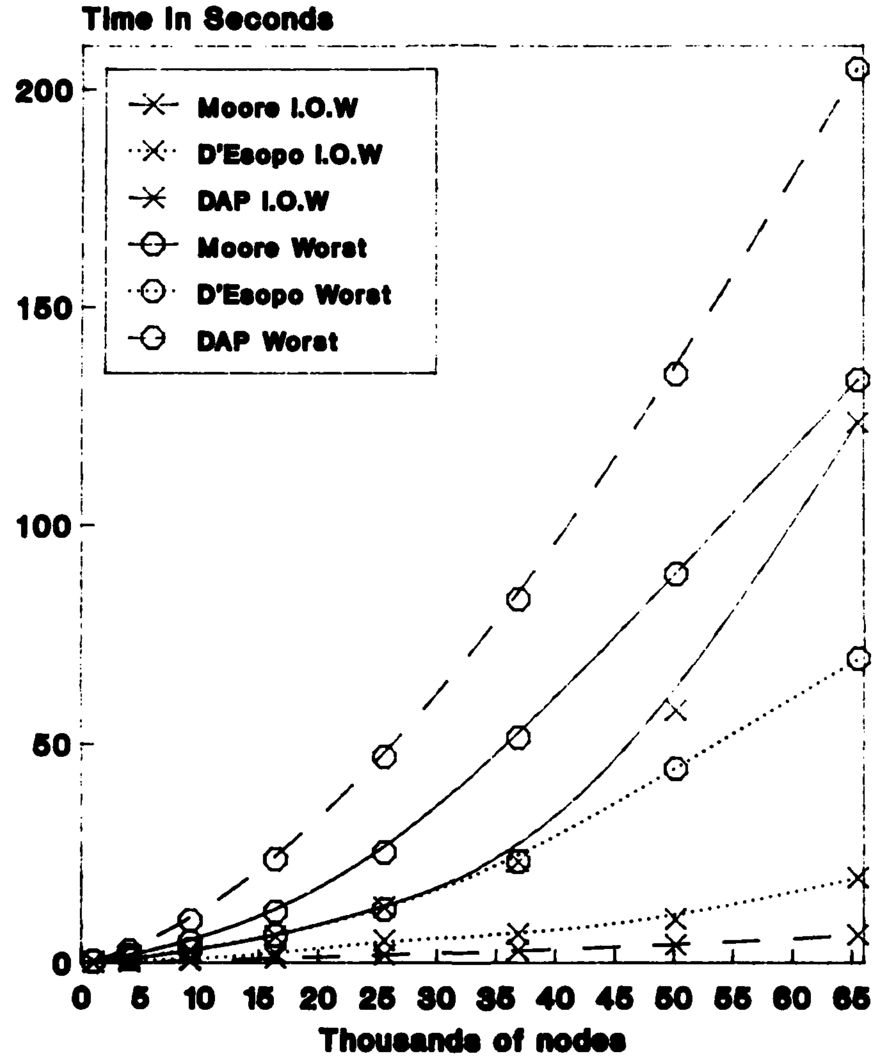
Number of nodes	Type of algorithm	Time in seconds	Number of comparisons	Number of alterations
1024	MR	0.11	13,884	3,975
	DS	0.12	13,760	3,975
	DAP	0.02	92,160	368,640
4096	MR	0.75	78,804	212,464
	DS	0.42	34,436	10,221
	DAP	0.17	524,288	1,347,584
9216	MR	2.70	247,988	70,324
	DS	0.80	82,256	24,793
	DAP	0.49	2,055,168	3,526,656
16384	MR	6.07	590,076	168,528
	DS	2.12	230,984	66,689
	DAP	1.05	5,242,880	7,225,344
25600	MR	12.7	1,231,620	353,872
	DS	5.1	521,676	145,403
	DAP	1.8	8,780,800	12,525,568
36864	MR	23.2	2,046,320	590,736
	DS	6.6	670,400	189,049
	DAP	2.8	14,856,192	18,853,888
50176	MR	57.7	3,191,448	920,243
	DS	10.0	951,672	267,673
	DAP	4.1	25,138,176	27,324,416
65536	MR	123.5	5,510,808	1,549,313
	DS	19.5	1,916,768	519,230
	DAP	6.4	49,086,464	4,142,818

B.2.2 Algorithms run using the worst case terrain data.

MR - Moore's algorithm
DS - D'Esopo's algorithm
DAP - Dynamic algorithm with active block (on the DAP)
Comparisons - Nodes checked for possible updating, (for the DAP this includes all the nodes in the block).
Alterations - Nodes that are updated, (for the DAP all the nodes in the selected blocks are included even if not all are changed).

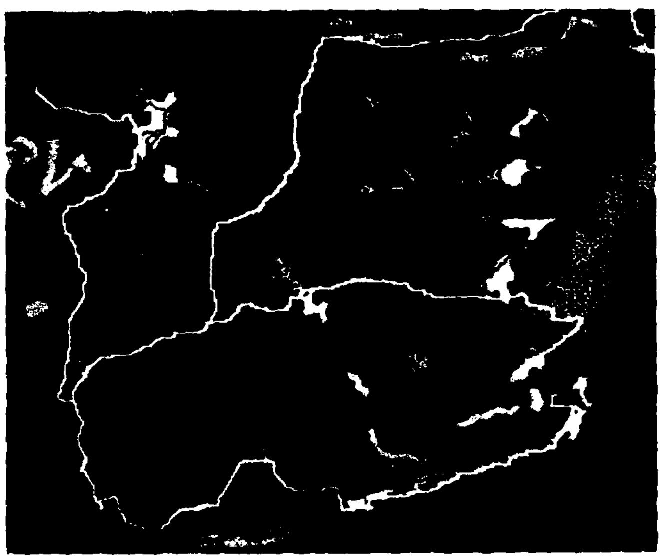
Number of nodes	Type of algorithm	Time in seconds	Number of comparisons	Number of alterations
1024	MR	0.19	20,384	5,313
	DS	0.13	14,780	4,032
	DAP	0.27	540,672	2,162,688
4096	MR	1.56	151,300	37,761
	DS	0.83	102,780	27,136
	DAP	2.70	8,450,048	17,113,088
9216	MR	4.86	478,944	121,921
	DS	2.89	329,532	85,696
	DAP	9.59	42,542,080	57,435.920
16384	MR	11.6	1,113,728	282,369
	DS	6.2	760,572	196,096
	DAP	23.6	133,644,288	135,684,096
25600	MR	25.2	2,149,920	543,681
	DS	12.2	1,461,436	374,720
	DAP	47.0	325,529,600	264,470,528
36864	MR	51.4	3,685,824	930,433
	DS	23,2	2,497,660	637,952
	DAP	83.0	673,984,512	456,368,128
50176	MR	88.8	5,819,968	1,467,201
	DS	44.3	3,934,780	1,002,176
	DAP	134.5	1,247,275,008	723,959,808
65536	MR	133.2	8,649,984	2,178,561
	DS	69.5	5,838,332	1,483,776
	DAP	204.6	2,126,053,376	1,079,828,480

B.3 Run Times For The Algorithms.



APPENDIX C: Photographs from Mil-DAP Implementation

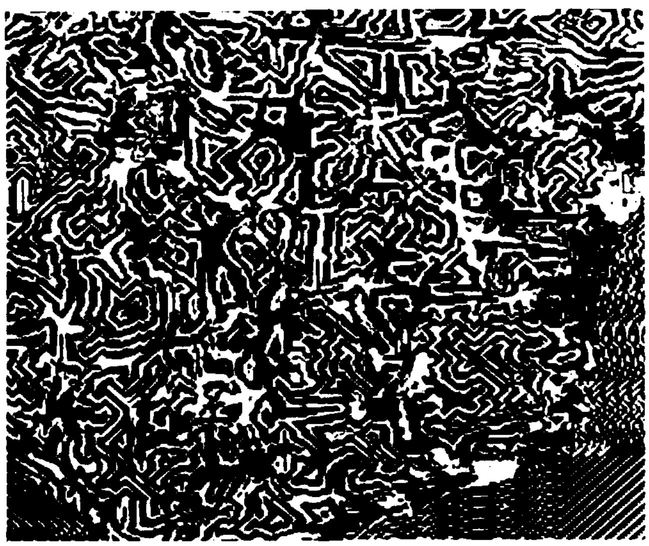
C.1: Routes selected for a Tank, start point near bottom left corner.



C.2: Routes selected for a car, start point near bottom left corner.

INTENTIONALLY
BLANK

C.3: Cost contour map for a tank, start point near bottom left corner.



C.4: Cost contour map for a car, start point near bottom left corner.

C.5: Progression of the wave of activity across the map, start point top left corner.

C.6: Progression of wavefront activity across the map, start point top left corner.

APPENDIX D.1.

```
      ENTRY SUBROUTINE BEST

      INTEGER*2 DISP(,,8,8)
      INTEGER*1 INCULT(,,8,8)
      CHARACTER ROUTE(,,8,8)
      INTEGER*2 X,Y,BIG,BX,BY,TX,TY
      INTEGER*2 BLK(,),VALUE(8),NEWS(,)
      INTEGER*2 DN(,,8,8),DS(,,8,8)
      INTEGER*2 DE(,,8,8),DW(,,8,8)
      INTEGER*4 N(2)
      LOGICAL MASK(,),CULTURE(,,8,8,8)
      LOGICAL ACTIVE(,),ROADS(,)

      COMMON /D/ DISP
      COMMON /S/ X,Y
      COMMON /R/ ROUTE
      COMMON /T/ TX,TY
      COMMON /CU/ INCULT
      COMMON /CO/ VALUE
      COMMON /N/ N
      COMMON /DN/DN
      COMMON /DS/DS
      COMMON /DE/DE

      EQUIVALENCE (CULTURE, INCULT)

C Copy uncorrupted map data from DS to DISP.
      DO 5 I=1,8
      DO 5 J=1,8

      DISP(,,I,J)=DS(,,I,J)

5     CONTINUE

C  Initialise values and calculate difference matrices

C  Define infinity!

      BIG=16129

      CALL CONVFV2(VALUE,8,1)

C  Get start point from tablet LOCATE procedure

      X=TX
      Y=TY

      ACTIVE(,)=.FALSE.

C  Calculate height difference and add travel weighting for move-cost
C  matrices of north, east, south and west moves.

      DO 15 I=1,8
      DO 15 J=1,8

      DN(,,I,J)=(ABS(DISP(,,I,J)-SHSP(DISP(,,I,J))))
      DN(,,I,J)=DN(,,I,J)*VALUE(2)+VALUE(1)
      IF (I.GT.1) DN(1,,I,J)=(ABS(DISP(1,,I,J)-DISP(32,,I-1,J)))
      IF (I.GT.1) DN(1,,I,J)=DN(1,,I,J)*VALUE(2)+VALUE(1)

      DE(,,I,J)=(ABS(DISP(,,I,J)-SHWP(DISP(,,I,J))))
      DE(,,I,J)=DE(,,I,J)*VALUE(2)+VALUE(1)
      IF (J.LT.8) DE(,32,I,J)=(ABS(DISP(,32,I,J)-DISP(,1,I,J+1)))
      IF (J.LT.8) DE(,32,I,J)=DE(,32,I,J)*VALUE(2)+VALUE(1)
```

```
      DS(,,I,J)=(ABS(DISP(,,I,J)-SHNP(DISP(,,I,J))))
      DS(,,I,J)=DS(,,I,J)*VALUE(2)+VALUE(1)
      IF (I.LT.8) DS(32,,I,J)=(ABS(DISP(32,,I,J)-DISP(1,,I+1,J)))
      IF (I.LT.8) DS(32,,I,J)=DS(32,,I,J)*VALUE(2)+VALUE(1)

      DW(,,I,J)=(ABS(DISP(,,I,J)-SHEP(DISP(,,I,J))))
      DW(,,I,J)=DW(,,I,J)*VALUE(2)+VALUE(1)
      IF (J.GT.1) DW(,1,I,J)=(ABS(DISP(,1,I,J)-DISP(,32,I,J-1)))
      IF (J.GT.1) DW(,1,I,J)=DW(,1,I,J)*VALUE(2)+VALUE(1)


C Make roads thicker so diagonals can be included.
      ROADS= SHNP(CULTURE(,,8,I,J))
      IF (I.LT.8) ROADS(32,)= CULTURE(,,8,I+1,J)
      CULTURE(ROADS,8,I,J)=.TRUE.

C  Add to move-cost matrices the cost of culture details,
C  trees,buildings,water and railway tracks.
      DO 10 K=3,7

C  Set up mask so if any culture detail consides with a road it is ignored.
      MASK =CULTURE(,,K,I,J).AND.(.NOT.CULTURE(,,8,I,J))

      DN(MASK,I,J)=DN(,,I,J)+VALUE(K)
      DS(MASK,I,J)=DS(,,I,J)+VALUE(K)

      DE(MASK,I,J)=DE(,,I,J)+VALUE(K)
      DW(MASK,I,J)=DW(,,I,J)+VALUE(K)

10    CONTINUE

C  Add to move-cost matrices the cost of not travelling by road
      MASK =.NOT.CULTURE(,,8,I,J)

      DN(MASK,I,J)=DN(,,I,J)+VALUE(8)
      DS(MASK,I,J)=DS(,,I,J)+VALUE(8)

      DE(MASK,I,J)=DE(,,I,J)+VALUE(8)
      DW(MASK,I,J)=DW(,,I,J)+VALUE(8)

15    CONTINUE

C  Inititialise cumulative cost matrices to "inifinity"

      DO 20 I=1,8
      DO 20 J=1,8

      DISP(,,I,J)=BIG

20    CONTINUE

C  Start clock
      CALL PRETIMER

C  Set cumulative cost at start point to zero.

      BX=(X-1)/32
      BY=(Y-1)/32

      ACTIVE(BY+1,BX+1)=.TRUE.

      DISP(Y-BY*32,X-BX*32,BY+1,BX+1)=0
C ---------------------------------------------------------------

C   Main loop calculates best moves from each square.
```

```
      N(1)=0
      N(2)=0
c     N = 0
c30    N = N + 1
30    CONTINUE

      DO 40 I=1,8
      DO 40 J=1,8
      N(1) = N(1) + 1

      IF (.NOT.ACTIVE(I,J)) GOTO 40
      N(2) = N(2) + 1
      NEWS(,)=DISP(,,I,J)

C--------------------------------------------
C NORTH
C BLK is made equal to current cost

      BLK= SHSP(DISP(,,I,J))
      IF (I.GT.1) BLK(1,)= DISP(32,,I-1,J)
      IF (I.EQ.1) BLK(1,)= BIG

C Cost of moving north added

      BLK= BLK + DN(,,I,J)


C Make logical of overlay where cost is less

      MASK= BLK.LT.NEWS(,)

C Copy new values onto old costs

      NEWS(MASK)= BLK

C Add to mask of back markers to indicate from where the route come.

      ROUTE(MASK,I,J)= 'S'

C--------------------------------------------
C EAST
      BLK= SHWP(DISP(,,I,J))
      IF (J.LT.8) BLK(,32)= DISP(,1,I,J+1)
      IF (J.EQ.8) BLK(,32)= BIG
      BLK= BLK + DE(,,I,J)
      MASK= BLK.LT.NEWS(,)
      NEWS(MASK)= BLK
      ROUTE(MASK,I,J)= 'W'

C ------------------------------------------------------
C SOUTH
      BLK= SHNP(DISP(,,I,J))
      IF (I.LT.8) BLK(32,)= DISP(1,,I+1,J)
      IF (I.EQ.8) BLK(32,)= BIG
      BLK= BLK + DS(,,I,J)
      MASK= BLK.LT.NEWS(,)
      NEWS(MASK)= BLK
      ROUTE(MASK,I,J)= 'N'

C--------------------------------------------
C WEST
      BLK= SHEP(DISP(,,I,J))
      IF (J.GT.1) BLK(,1)= DISP(,32,I,J-1)
      IF (J.EQ.1) BLK(,1)= BIG
      BLK= BLK + DW(,,I,J)
```

```
      MASK= BLK.LT.NEWS(,)
      NEWS(MASK)= BLK
      ROUTE(MASK,I,J)= 'E'

C------------------------------------------------

C Which 32x32 blocks are active
      ACTIVE(I,J) = .NOT.(ALL(NEWS(,).EQ.DISP(,,I,J)))

      IF ((ANY(DISP(1,,I,J).NE.NEWS(1,))).AND.(I.GT.1))
     1  ACTIVE(I-1,J) = .TRUE.

      IF ((ANY(DISP(32,,I,J).NE.NEWS(32,))).AND.(I.LT.8))
     1  ACTIVE(I+1,J) = .TRUE.

      IF ((ANY(DISP(,1,I,J).NE.NEWS(,1))).AND.(J.GT.1))
     1  ACTIVE(I,J-1) = .TRUE.

      IF ((ANY(DISP(,32,I,J).NE.NEWS(,32))).AND.(J.LT.8))
     1  ACTIVE(I,J+1) = .TRUE.

C Copy new set of costs into old cost matrix
      DISP(,,I,J)=NEWS(,)

40    CONTINUE

C If routing not finished repeat process
      IF (ANY(ACTIVE(,))) GOTO 30

C Stop clock
      CALL POSTTIMER

60    CONTINUE

      CALL CONVVF4(N,2,1)

      DO 100 I=1,8
      DO 100 J=1,8
      DE(,,I,J)=DISP(,,I,J)
100   CONTINUE

      RETURN
      END
```

APPENDIX D.2.

```
      SUBROUTINE MOORE
C
       IMPLICIT INTEGER*4 (A-Z)
       INTEGER V,L2
C
       COMMON /NETW/ A(66000),B(264000),S,NA,NH,SQ,LT
       COMMON /TIME/ T(264000)
       COMMON /LABELS/ L1(66000),L2(66000)
       COMMON /QUEUE/ Q(66000)
C
C MOORE ALGORITHM
C
C START BY ASSIGNING LABELS TO ALL NODES OF THE FORM (L1(R),L2(R),
C WHERE:-
C      L1(R) = BLANK FOR ALL NODES R=1,2,.....
C              N IS THE HIGHEST NODE NUMBER.
C              L1(R) WILL BE USED TO STORE THE BACK NODE NUMBER
C              WHICH WILL ALLOW THE PATH BETWEEN THE ORIGIN AND
C              DESTINATION TO BE RETRACED.
C              IN THE PROGRAM L1(R) IS SET TO 'NH' WHICH IS THE
C              HIGHEST NODE NUMBER PLUS ONE, I.E. A NON EXISTENT
C              NODE NUMBER.
C      L2(R) = INFINITY (9999999) FOR ALL NODES 1,2,.....N
C
C      Q(R)  = 0 FOR WHOLE QUEUE.
C
C      IF THE ORIGIN NODE IS 'S', SET L1(S)=0, L2(S)=0 AND Q(1)=S.
C
C      SEARCH FOR ANY LINK (FROM NODE 'A' TO NODE 'B' =(A,B))
C      SUCH THAT
C
C                 L2(A) + T(A,B) < L2(B)
C
C      WHERE T(A,B) IS THE TRAVEL TIME FROM NODE 'A' TO NODE 'B'
C      ON LINK (A,B). IF SUCH A LINK IS FOUND CHANGE THE LABELS
C      ATTACHED TO NODE 'B' TO
C
C                 L1(B) = A
C             AND L2(B) = L2(A) + T(A,B)
C
C      REPEAT THE PROCESS UNTIL NO SUCH LINK IS FOUND, THEN THE
C      ALGORITHM TERMINATES.
C
       WRITE(6,*)' MOORE '
       NH=NA+1
       F=0
       G=0
       QS=0
       QD=0
       QE=NH
       DO 420 I=1,NA
       L1(I)=NH
       L2(I)=9999999
       Q(I)=0
 420   CONTINUE
C -------------------- SET ORIGIN LABELS
       L1(S)=0
       L2(S)=0
       Q(1)=S

C
C -------------------- SEARCH ALL LINKS FOR L2(A)+T(A,B)<L2(B).
C                      START WITH LINKS FROM THE ORIGIN NODE 'S'.
C                      IF A 'B' NODE LABEL IS CHANGED THE NODE IS
```

```
C                   PUT ON A QUEUE.THE NODE IS ONLY PUT ON THE
C                   END OF THE QUEUE IF IT IS NOT ON THE QUEUE
C                   ALREADY.
C                   'R' IS THE CURRENT 'A' NODE.
C                   A CIRCULAR QUEUE IS USED, MAXIMUM LENGH NA.
C                   Q(QU) IS THE POINTER FOR THE FRONT OF THE
C                   QUEUE AND Q(QE) THE END OF THE QUEUE.
C                   'IA' TO 'LA' ARE THE LOCATIONS OF THE LINKS
C                   FROM THE NODE 'R' IN THE ARRAY 'B',ETC.
580     CONTINUE
        QU=JMOD(QS,NA)+1
        QS=QS+1
        R=Q(QU)
        L1(R)=-L1(R)
        TA=L2(R)
        IA=A(R)
        LA=A(R+1)-1
        DO 720 K=IA,LA
        P=B(K)
        TB=TA+T(K)
        F=F+1
        IF (L2(P).LE.TB) GOTO 720
        G=G+1
        IF (L1(P).LT.0) GOTO 710
C QE is the end marker for the circular queue(Q).
        QD=QD+1
        QE=JMOD(QD,NA)+1
        Q(QE)=P
 710    L1(P)=-R
        L2(P)=TB
 720    CONTINUE
        IF (QU.EQ.QE) GOTO 940
        GOTO 580
 940    CONTINUE
        WRITE(6,*)' NUMBER OF CHECKS FOR ALTERATIONS ',F
        WRITE(6,*)' NUMBER OF ALTERATIONS ',G
        WRITE(6,*)' L2(NA) ',L2(NA)

        RETURN
        END
```

APPENDIX D.3.

```
      SUBROUTINE DESOPO
C
      IMPLICIT INTEGER*4 (A-Z)
      INTEGER V,L2
C
      COMMON /NETW/ A(66000),B(264000),S,NA,NH,SQ,LT
      COMMON /TIME/ T(264000)
      COMMON /LABELS/ L1(66000),L2(66000)
      COMMON /QUEUE/ Q(66000)
C
C DESOPO ALGORITHM
C
C START BY ASSIGNING LABELS TO ALL NODES OF THE FORM (L1(R),L2(R),
C WHERE:-
C      L1(R) = BLANK FOR ALL NODES R=1,2,.....
C              N IS THE HIGHEST NODE NUMBER.
C              L1(R) WILL BE USED TO STORE THE BACK NODE NUMBER
C              WHICH WILL ALLOW THE PATH BETWEEN THE ORIGIN AND
C              DESTINATION TO BE RETRACED.
C              IN THE PROGRAM L1(R) IS SET TO 'NH' WHICH IS THE
C              HIGHEST NODE NUMBER PLUS ONE, I.E. A NON EXISTENT
C              NODE NUMBER.
C      L2(R) = INFINITY (9999999) FOR ALL NODES 1,2,.....N
C
C      Q(R)  = 0 FOR WHOLE QUEUE.
C
C      IF THE ORIGIN NODE IS 'S', SET L1(S)=0, L2(S)=0 AND Q(1)=S.
C
C      SEARCH FOR ANY LINK (FROM NODE 'A' TO NODE 'B' =(A,B))
C      SUCH THAT
C
C                 L2(A) + T(A,B) < L2(B)
C
C      WHERE T(A,B) IS THE TRAVEL TIME FROM NODE 'A' TO NODE 'B'
C      ON LINK (A,B). IF SUCH A LINK IS FOUND CHANGE THE LABELS
C      ATTACHED TO NODE 'B' TO
C
C                 L1(B) = A
C             AND L2(B) = L2(A) + T(A,B)
C
C      REPEAT THE PROCESS UNTIL NO SUCH LINK IS FOUND WHEN THE
C      ALGORITHM TERMINATES.
C
      WRITE(6,*)' DESOPO '
      NH=NA+1
      F=0
      G=0
      QS=0
      QD=0
      QE=NH
      DO 420 I=1,NA
      L1(I)=NH
      L2(I)=9999999
      Q(I)=0
 420  CONTINUE
C -------------------- SET ORIGIN LABELS
      L1(S)=0
      L2(S)=0
      Q(1)=S

C
C -------------------- SEARCH ALL LINKS FOR L2(A)+T(A,B)<L2(B).
C                      START WITH LINKS FROM THE ORIGIN NODE 'S'.
```

```
C                         IF A 'B' NODE LABEL IS CHANGED THE NODE IS
C                         PUT ON A QUEUE.THE NODE IS ONLY PUT ON THE
C                         QUEUE IF IT IS NOT AT PRESENT ON THE QUEUE.
C                         IF THE NODE  HAS  PREVIOUSLY BEEN  ON  THE
C                         QUEUE IT IS PUT AT THE FRONT OF THE QUEUE.
C                         'R' IS THE CURRENT 'A' NODE.
C                         A CIRCULAR QUEUE IS USED, MAXIMUM LENGH NA.
C                         Q(QU) IS THE POINTER FOR THE FRONT OF THE
C                         QUEUE AND Q(QE) THE END OF THE QUEUE.
C                         'IA' TO 'LA' ARE THE LOCATIONS OF THE LINKS
C                         FROM THE NODE 'R' IN THE ARRAY 'B',ETC.
580     CONTINUE
        QU=JMOD(QS,NA)+1
        QS=QS+1
        R=Q(QU)
        L1(R)=-L1(R)
        TA=L2(R)
        IA=A(R)
        LA=A(R+1)-1
        DO 720 K=IA,LA
        P=B(K)
        TB=TA+T(K)
        F=F+1
        IF (L2(P).LE.TB) GOTO 720
        G=G+1
C QE is the end marker for the circular queue(Q).
        IF (L1(P).LT.0) GOTO 710
        IF (L1(P).LT.NH) GOTO 700
        QD=QD+1
        QE=JMOD(QD,NA)+1
        Q(QE)=P
        GOTO 710
 700    QS=QS-1
        QU=JMOD(QS,NA)+1
        Q(QU)=P
 710    L1(P)=-R
        L2(P)=TB
 720    CONTINUE
        IF (QU.EQ.QE) GOTO 940
        GOTO 580
 940    CONTINUE
        WRITE(6,*)' NUMBER OF CHECKS FOR ALTERATIONS ',F
        WRITE(6,*)' NUMBER OF ALTERATIONS ',G
        WRITE(6,*)' L2(NA) ',L2(NA)

        RETURN
        END
```

DOCUMENT CONTROL SHEET

Overall security classification of sheet .... UNCLASSIFIED ....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification
	MEMO 4299		U/C

5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location
7784000	ROYAL SIGNALS AND RADAR ESTABLISHMENT ST ANDREWS ROAD, GREAT MALVERN, WORCS WR14 3PS
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location

7. Title

COMPUTING OPTIMUM ROUTES ACROSS TERRAIN USING DAP

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
HAYNES, P	PACKER, C	ROBERTS, J B G	07.1989	33

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement

UNLIMITED

Descriptors (or keywords)

continue on separate piece of paper

Abstract

Dynamic programming techniques provide quantitative ways for determining optimum routes across complex terrain for various assumed categories of vehicle, taking account of gradients, the presence or absence of roads and obstacles such as rivers, urban areas, woods etc. We have investigated the effectiveness of a highly parallel SIMD computer. (Mil-DAP) to this problem. We conclude that, although impressive computation speeds can be obtained on high resolution digital maps with Mil-DAP, the conflict between exploiting high order SIMD parallelism and minimising the number of necessary computation operations is such that in this case Mil-DAP does not command a dominating advantage over a conventional uniprocessor.

S80/48